Homework 2 — ML core language — assigned Tuesday 27 January — due Wednesday 4 February

Reading assignment

Read Chapters 1, 2, and 3 of *ML for the Working Programmer*.

2.1 Integers (5pts)

Define a function called *cube*, of type $int \rightarrow int$, which returns the integer which is the cube of the integer it is applied to.

2.2 List types (5pts)

Write a function *swapl* that takes a list of pairs as argument and returns a list of pairs in which the elements of each pair are swapped. Specify its type.

2.3 List types and higher-order functions (5pts)

Write a function *map2* that applies a function to all elements in all element lists in a list of lists. Specify its type.

2.4 Using lists for arithmetic: writing recursive functions over lists (40pts)

Numerals can be represented as lists of integers. For instance, decimal numerals can be expressed as lists of integers from 0 to 9. In this representation, the integer 12345678901234567890 would be represented as the ML list `[1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0]: int list`.

Write the following functions:

- (10pts) `makeLongInt: int -> int -> int list`, such that `makeLongInt` *r n* computes the list representation of the integer *n* in radix *r*. You can assume that $n \geq 0$, and that $r > 1$. For example, `makeLongInt 10 123` should evaluate to `[1,2,3]`.
- (10pts) `evaluateLongInt: int -> int list -> int`, such that `evaluateLongInt` *r l* computes an integer corresponding to the value of list *l*, which uses radix *r*. You can assume that *l* is a valid list for radix *r*, and the value of the list is small enough to fit into an ML `int`. For example, `evaluateLongInt 10 [1,2,3]` should evaluate to `123`.
- (20pts) `addLongInts: int -> (int list * int list) -> int list`, such that `addLongInts` *r* (a,b) computes the sum of the nonnegative integers given by lists *a* and *b*; all three lists use radix *r*. For example, `addLongInts 10 ([1,2,3], [1])` should evaluate to `[1,2,4]`. Important: the argument lists *a* and *b* can represent arbitrarily large integers.

Hint: you need to learn how to control the top level of SML/NJ so that it does not truncate very long lists when it prints them.

2.5 Drawing: writing recursive functions over lists; manipulating strings (45pts)

In this exercise, we develop some simple tools for drawing.

A drawing is just a line drawing consisting of some number of polygons. A polygon is given as a list of vertices, and a vertex is simply a pair of real numbers for the *x* and *y* coordinates.

For instance,

```
[[(100.0,100.0),(100.0,200.0),(200.0,100.0)],
[(150.0,150.0),(150.0,200.0),(200.0,200.0),(200.0,150.0)]]
```

is an internal representation in ML of a drawing consisting of a triangle and a square.

Your task is to convert such a representation of a drawing into a simple page description in the PostScript language. Specifically, you are to write an ML function `makeCommand: (real * real) list list -> string`.

The result returned by `makeCommand` is an ML value of type string, which must contain valid PostScript commands for drawing the given polygons. When the SML/NJ top level prints this result, paste it by hand into a file *result.ps*, and then display it by running GhostScript: `gs result.ps`.

For instance, the expression

```
makeCommand [[(100.0,100.0),(100.0,200.0),(200.0,100.0)],
[(150.0,150.0),(150.0,200.0),(200.0,200.0),(200.0,150.0)]]
```

should evaluate to the string:

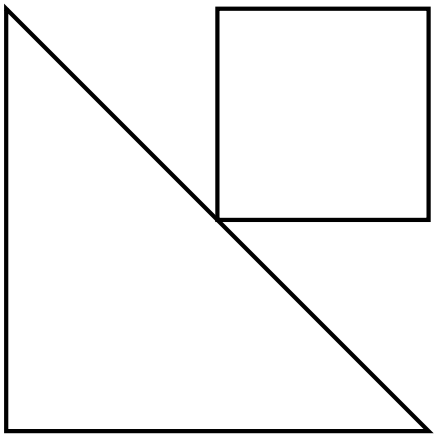
```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 100.0 100.0 200.0 200.0

100.0 100.0 moveto
100.0 200.0 lineto
200.0 100.0 lineto
closepath
stroke

150.0 150.0 moveto
150.0 200.0 lineto
200.0 200.0 lineto
200.0 150.0 lineto
closepath
stroke

showpage
%%EOF
```

which will be displayed as the following figure:



Note that the bounding box is the smallest upright rectangle such that no points of the drawing lie outside it; it is specified by giving the coordinates of its lower left and upper right corners, in our example $(100.0, 100.0)$ and $(200.0, 200.0)$.

The example shown here entirely suffices as a pattern to follow; however, if you would like to learn more about the PostScript language you can follow the links on the course web page.

Hint: you need to learn how to control the top level of SML/NJ so that it does not truncate very long strings when it prints them.

How to turn in

Make sure that you have thoroughly tested your code, and include all your test runs!

Turn in your code by running

~clint/handin your-file

on a regular UNM CS machine. You should use whatever filename is appropriate in place of your-file.

Include the following statement with your submission, signed and dated:
I pledge my honor that in the preparation of this assignment I have complied with the University of New Mexico Board of Regents' Policy Manual, including Section 4.8, Academic Dishonesty.